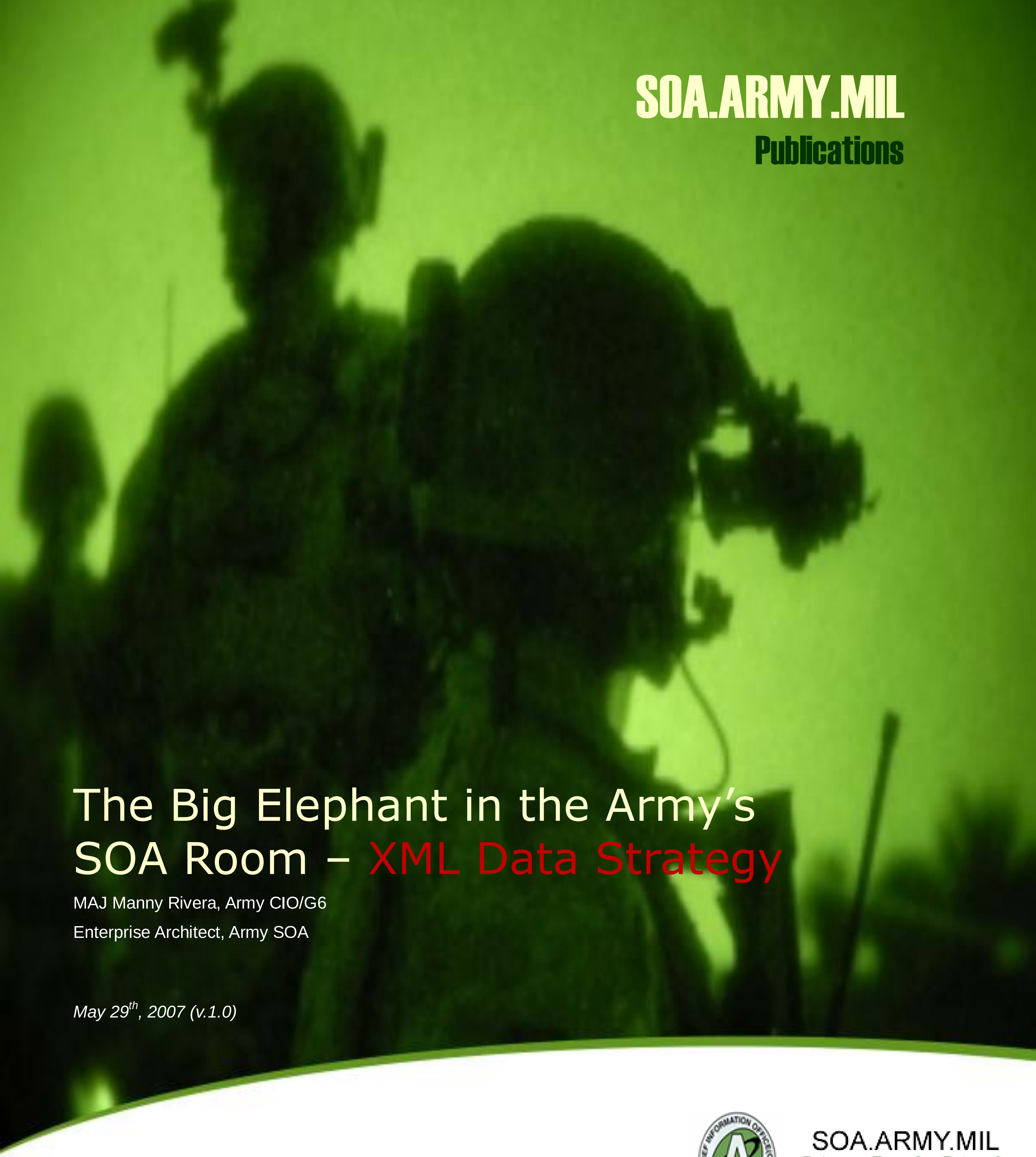


SOA.ARMY.MIL

Publications

The Big Elephant in the Army's SOA Room – XML Data Strategy

MAJ Manny Rivera, Army CIO/G6

Enterprise Architect, Army SOA

May 29th, 2007 (v.1.0)

SOA.ARMY.MIL

CIO/G6

Table of Contents

Executive Summary

The Army's SOA strategy has eight distinct components working together orchestrating the composition and choreography of a very complex system. At the core of the system are objects represented as Web services. Many SOA architects believe everything begins with Web services, and therefore most of the SOA efforts are directed at the service orientation of these Web services, which includes:

- Loose coupling
- Service Contract
- Autonomy
- Abstraction
- Reusability
- Composability
- Statelessness
- Discoverability

The Army should focus on service orientation and the above mentioned characteristics, but before we can get there from where we currently are, we must first focus on what got us here in the first place; 'XML". In fact, everything in the world of SOA begins with XML. It is the standard from which multiple supplementary standards have evolved to form the core data representation (model) architecture. It is this core set of standards that has fueled the creation of the many Web services specifications that are now driving SOA: SOAP, WSDL, XSD and the list goes on and on.

The Army and most DoD service components have put incredible amount of time identifying an Enterprise Data model (singular) and strategy, but there are challenges with a single data model approach. A single data model capable of spanning all Army Mission Areas (MA) would be very large, inflexible and difficult to maintain. This can put huge burdens on developed Web services, therefore, developers will be forced to find cleaner techniques, which in turn leads to inconsistencies and interoperability problems.

With one super data model to pull from, it makes it nearly impossible for validation and transportation; especially in cases where a Web service only requires a couple of elements to be validated for usage. At the end of the day, the Army should recognize that it is not about identifying super models, but, about the modularity, structure and validation of XML Data for usage by services in the background.

XML is incredibly modular, and the use of XML within Web services allows for the consumption of XML data from any model; JC3IEDM, GML, or a DoD agreed upon common standard.

This document will focus on standardizing the manner in which the Army's SOA represents, validates and process "Data" as it travels throughout application environments (both within and between Web services) and it lays the groundwork for a robust, optimized and interoperable SOA.

Key concepts covered in this document are: XML namespace, XML Schemas (XSD), SOAP, WSDL and Web services.

Avoiding Confusion by using XML Namespace

As illustrated in *figure 1*, XML Data Strategy cuts across all Mission Areas (MAs). Because Data is represented differently in the MAs, it is very important for Web services to know the precise owner of particular data elements in the Enterprise.

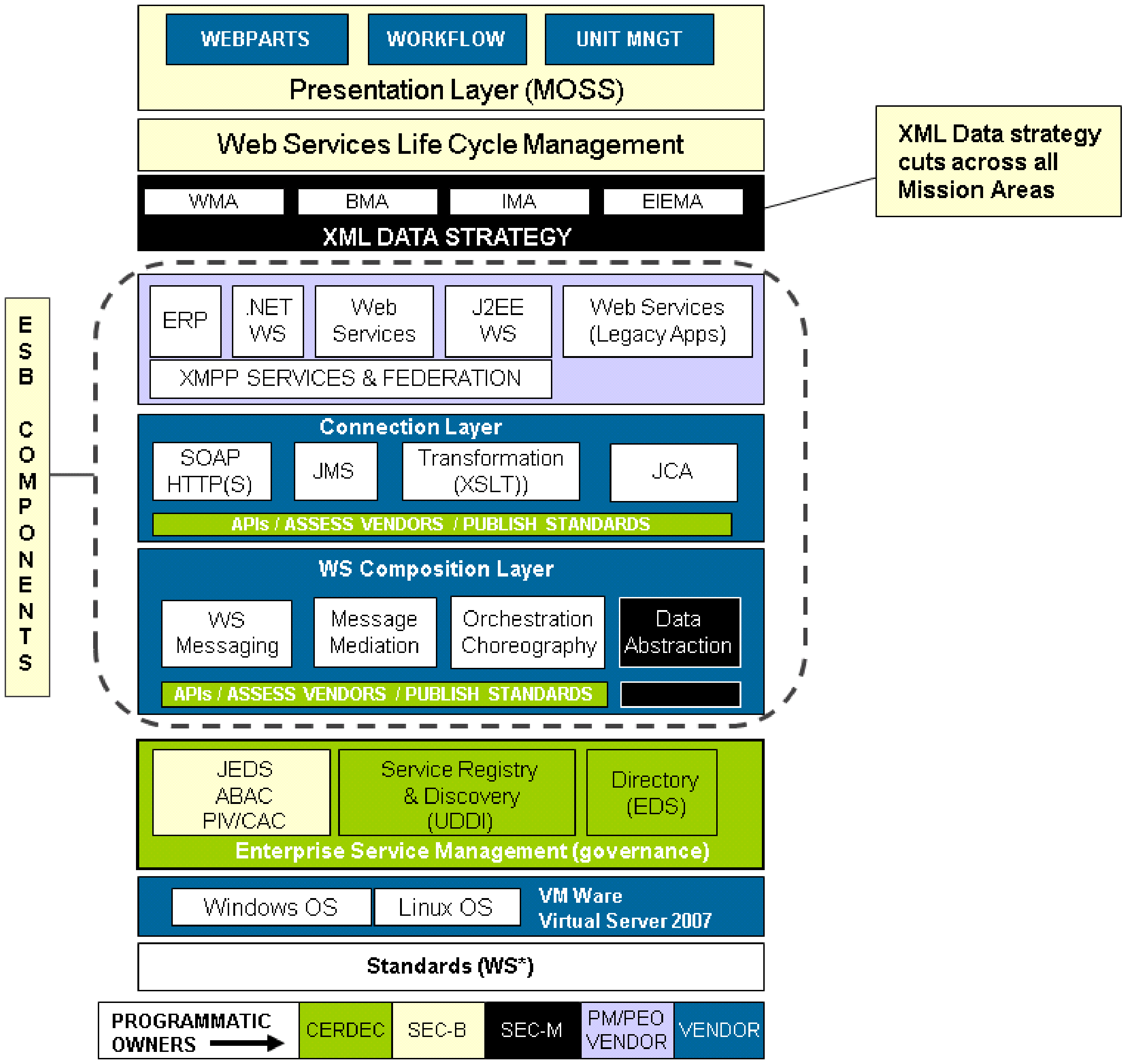

WEBPARTS
WORKFLOW
UNIT MNGT
Presentation Layer (MOSS)
Web Services Life Cycle Management
WMA
BMA
IMA
EIEMA
XML DATA STRATEGY
XML Data strategy cuts across all Mission Areas
ESB COMPONENTS
ERP
.NET WS
Web Services
J2EE WS
Web Services (Legacy Apps)
XMPP SERVICES & FEDERATION
Connection Layer
SOAP HTTP(S)
JMS
Transformation (XSLT))
JCA
APIs / ASSESS VENDORS / PUBLISH STANDARDS
WS Composition Layer
WS Messaging
Message Mediation
Orchestration Choreography
Data Abstraction
APIs / ASSESS VENDORS / PUBLISH STANDARDS
JEDS ABAC PIV/CAC
Service Registry & Discovery (UDDI)
Directory (EDS)
Enterprise Service Management (governance)
Windows OS
Linux OS
VM Ware Virtual Server 2007
Standards (WS*)
PROGRAMMATIC OWNERS
CERDEC
SEC-B
SEC-M
PM/PEO VENDOR
VENDOR

Figure 1, SOA Foundation Framework

The purpose of an XML namespace is to allow the deployment of XML vocabularies (in which element and attribute names are defined) in a global environment and to reduce the risk of name collisions in a given document when vocabularies are combined. For example, the JC3IEDM and GML models both may define an element named *position*. Although XML data from different formats such as JC3IEDM and GML can be combined in a single document, in this case there could be ambiguity about which *position* element was intended. XML Namespaces provide a method to avoid element name conflicts by taking advantage of existing systems for allocating globally scoped names.

XML Namespace is all about Scope

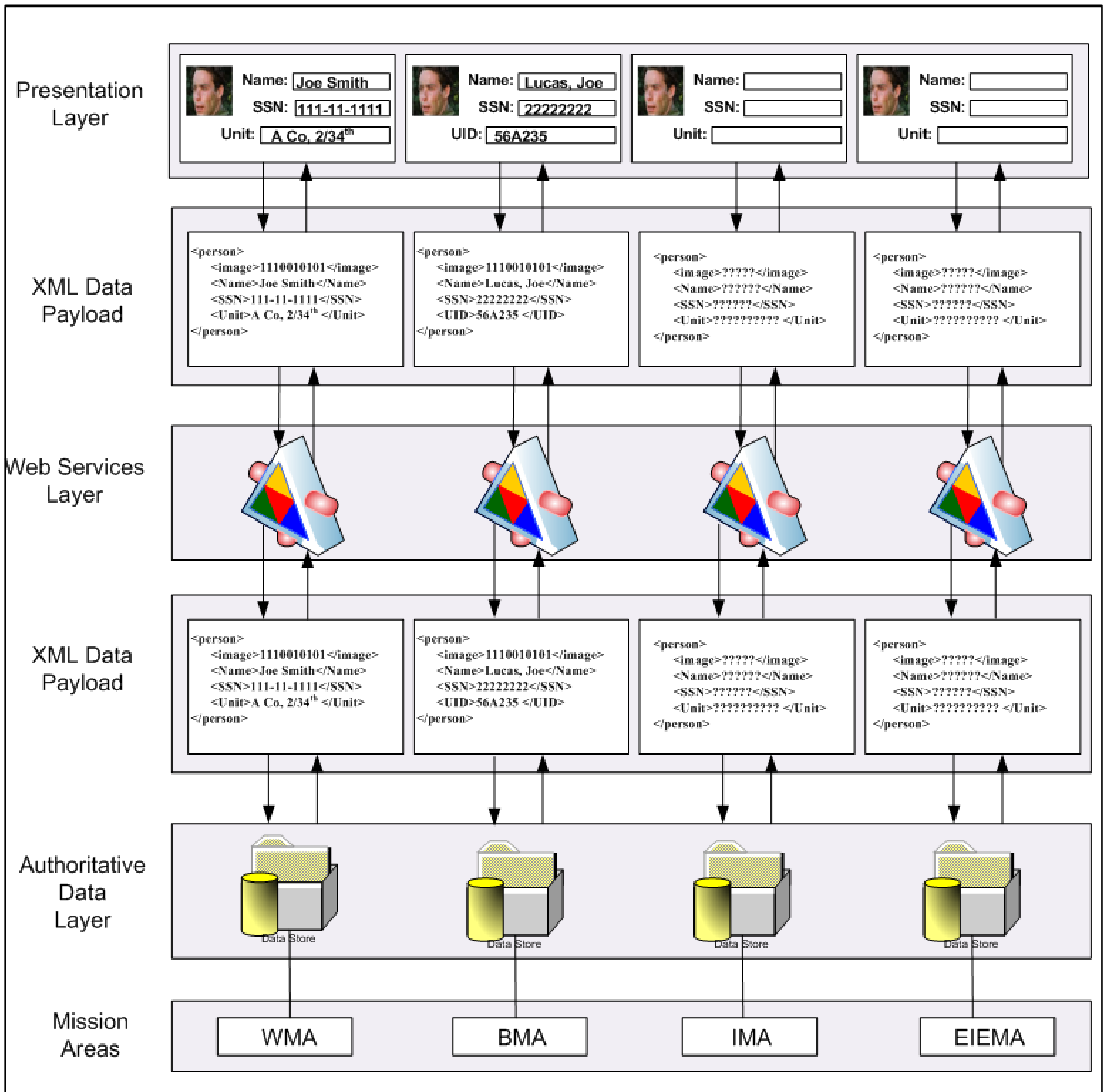
XML Data representation is not a problem when the data is scoped within its own domain. For example, here are two distinct XML data documents (payloads):

```
<person>
    <image>1110010101</image>
    <Name>Joe Smith</Name>
    <SSN>111-11-1111</SSN>
    <Unit>A Co, 2/34th</Unit>
</person>
```

```
<person>
    <image>1110010101</image>
    <Name>Lucas, Joe</Name>
    <SSN>222222222</SSN>
    <UID>56A235</UID>
</person>
```

The top XML data structure is the representation of a *person*, and it includes an *image, name, SSN and Unit* as its elements. This data structure could easily belong to a model inside the WMA. The bottom XML data structure is also the representation of a *person*, and it includes some of the elements from the top. However, instead of *Unit* as an element, it has *UID;* also the way *Name* and *SSN* are represented is different in the second one. The second XML data structure could easily belong to a model in the BMA.

As long as the XML data is one Web service to one MA (one-to-one), no conflict exist. As *figure 2* illustrates, XML data payloads can flow from MAs[1] through authoritative sources and Web services with no problem. The conflict happens when a Web service needs data



```
<person>
    <image>1110010101</image>
    <Name>Joe Smith</Name>
    <SSN>111-11-1111</SSN>
    <Unit>A Co, 2/34th </Unit>
</person>
```

```
<person>
    <image>1110010101</image>
    <Name>Lucas, Joe</Name>
    <SSN>22222222</SSN>
    <UID>56A235 </UID>
</person>
```

```
<person>
    <image>?????</image>
    <Name>??????</Name>
    <SSN>??????</SSN>
    <Unit>?????????? </Unit>
</person>
```

```
<person>
    <image>?????</image>
    <Name>??????</Name>
    <SSN>??????</SSN>
    <Unit>?????????? </Unit>
</person>
```

```
<person>
    <image>1110010101</image>
    <Name>Joe Smith</Name>
    <SSN>111-11-1111</SSN>
    <Unit>A Co, 2/34th </Unit>
</person>
```

```
<person>
    <image>1110010101</image>
    <Name>Lucas, Joe</Name>
    <SSN>22222222</SSN>
    <UID>56A235 </UID>
</person>
```

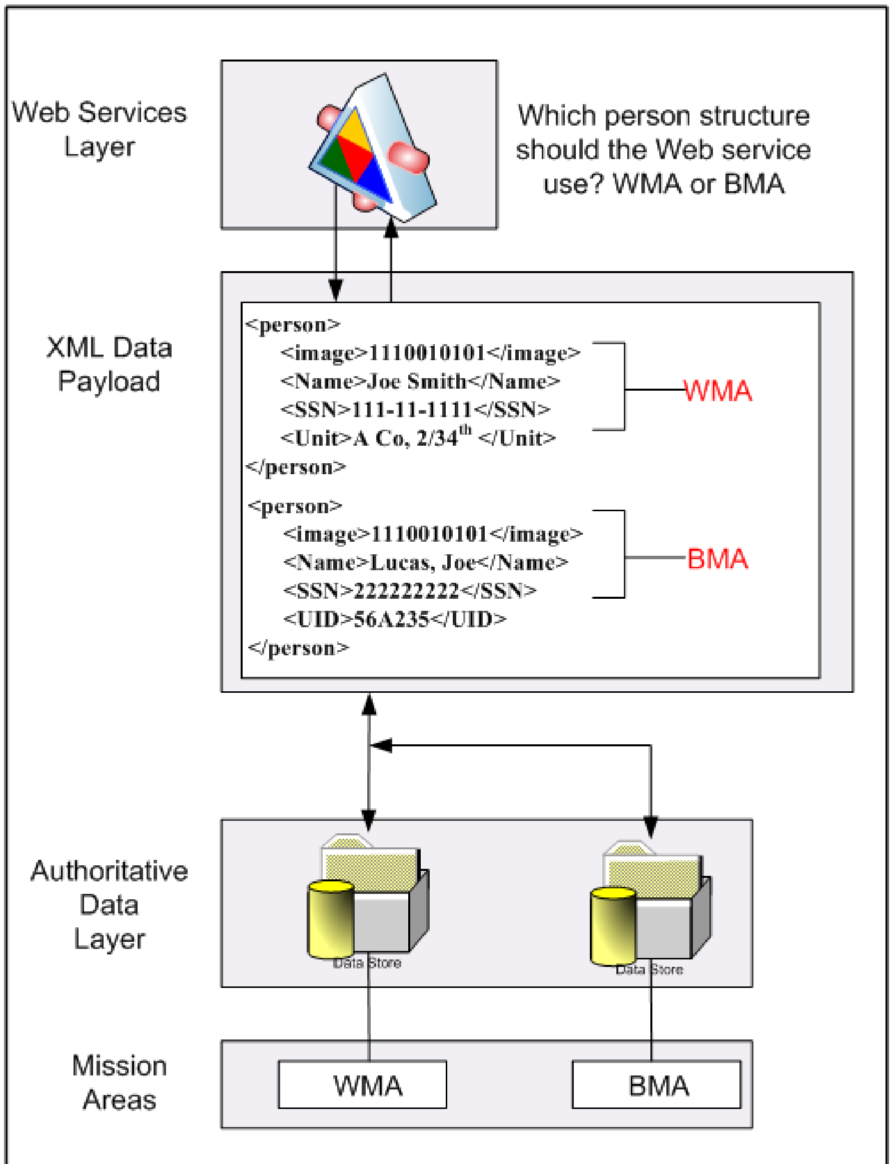
```
<person>
    <image>?????</image>
    <Name>??????</Name>
    <SSN>??????</SSN>
    <Unit>?????????? </Unit>
</person>
```

```
<person>
    <image>?????</image>
    <Name>??????</Name>
    <SSN>??????</SSN>
    <Unit>?????????? </Unit>
</person>
```

Figure 2. Data mapping through Web services (one-to-one)

[1] Although technically correct, XML data payloads can flow from any domain or system, we are representing it notionally as a top level (MA) structure for demonstration purposes.

from multiple places (or Mission Areas in our example); a very common occurrence in the SOA world. When this happens, as illustrated in *figure 3*, conflict exists and the Web service is not able to de-conflict the data structures; scope is no longer local to one MA, it changed to one Web service to many MAs (one-to-many).



```
<person>
    <image>1110010101</image>
    <Name>Joe Smith</Name>
    <SSN>111-11-1111</SSN>
    <Unit>A Co, 2/34th </Unit>
</person>
<person>
    <image>1110010101</image>
    <Name>Lucas, Joe</Name>
    <SSN>222222222</SSN>
    <UID>56A235</UID>
</person>
```

Figure 2, Data mapping through Web services (one-to-many)

Element type names clash when two or more XML data models use the same element type name to represent different real-world ideas or values.

Because the community using XML is already large and will grow further, the possibility that element type names will clash in shared documents and models becomes very real. The likelihood of element name clashes also increases as more XML documents (payloads) are shared outside defined groups.

Clearly, a mechanism is needed to distinguish a person in the WMA from a person in the BMA. We need to be able to express in XML, similar distinctions. In XML, elements are distinguished by using "*namespaces*."

A concept similar to XML namespaces exists in some programming languages. For example, in Java, a particular class must have a unique name, to avoid ambiguity. In Java, a package provides a broad equivalent to the concept of XML namespaces. A class must have a name that is unique within a package. Classes in other packages might have the same class name, but because they are in a different package, there is no risk of confusion as the code is processed.

As in Java, each XML element in an XML namespace must have a unique name; otherwise, confusion can arise.

Defining the namespace

To clearly distinguish the *person* element here data modelers would need to provide more information about the element type name than simply *person*:

```
<person>
    <image>1110010101</image>
    <Name>Joe Smith</Name>
    <SSN>111-11-1111</SSN>
    <Unit>A Co, 2/34th</Unit>
</person>
```

from the *person* element here

```
<person>
    <image>1110010101</image>
    <Name>Lucas, Joe</Name>
    <SSN>222222222</SSN>
    <UID>56A235</UID>
</person>
```

The solution to this, in the XML standards, is a qualified name. This is often abbreviated as QName. A qualified name is an XML element type name that consists of two parts—a namespace prefix and a local part separated by the colon character (:).

Data modelers can distinguish elements using qualified names (QNames). Referring back to the earlier example, the WMA version of a *person* could use the QName ***wma:person***. This would distinguish it from possible QNames for the BMA *person* element types, perhaps expressed as ***bma:person***.

Each QName consists of a namespace prefix, a colon character, and a local part. The local part is what we have called the element type name in a non–namespace-aware document.

The closest analogy we can make for namespace is by using a person's last name: ***smith:person***. This Qname tells us: *smith* is the owner of an element called *person*. Now, every time we need to reference an element belonging to *smith*, we just initialize it with smith's last name, a colon and then the element smith owns.

To be fair, using QNames on their own may produce problems similar to those we are trying to avoid. For example, the ***wma:person*** element might itself lead to ambiguity. Are we referring to a WMA person in JC3IEDM data model or wma person in a WMA IED data model? Similarly, if we refer to ***bma:person*** elements, which of potentially many element types created by individual BMA models are we referring to?

The Army needs a more universal way to distinguish namespaces than simply using namespace prefixes alone. A potential solution to achieving unique identification of a namespace is to use a uniform resource identifier (URI) and mapping a namespace prefix to it.

URIs and Namespace

In XML namespaces, the namespace name is a uniform resource identifier (URI).

A URI is a potentially lengthy sequence of characters. For example, an approach is to create a document type for a particular structure of document. We could choose namespace URIs:

- http://soa.army.mil/schemas/wma
- http://soa.army.mil/schemas/bma
- http://soa.army.mil/schemas/ima
- http://soa.army.mil/schemas/eiema

Declaring a namespace involves associating the namespace prefix with the namespace URI. In XML, this is done by using a special type of attribute called a namespace declaration.

To be used in an XML document, a namespace must be declared. A namespace declaration is made in the start tag of the element to which it refers. A namespace declaration has a special structure. For most namespace declarations, the attribute name begins with the character sequence xmlns, followed by a colon and the namespace prefix. These are followed by an equal sign (=) and the namespace URI enclosed in a pair of double or single quotation marks:

xmlns:WMA="http://soa.army.mil/schemas/wma"

Here's the previous example using qualified namespaces:

```
<?xml version="1.0"?>
<wma:person xmlns:wma="http://soa.army.mil/schemas/wma"
            xmlns:bma="http://soa.army.mil/schemas/bma">

     <wma:image>1110010101</wma:image>
     <wma:Name>Joe Smith</wma:Name>
     <wma:SSN>111-11-1111</wma:SSN>
     <wma:Unit>A Co, 2/34th</wma:Unit>

  <bma:person>
     <bma:image>1110010101</bma:image>
     <bma:Name>Lucas, Joe</bma:Name>
     <bma:SSN>222222222</bma:SSN>
     <bma:UID>56A325</bma:UID>
```

```
   </bma:person >
</wma:person >
```

The XML code snippet from above disambiguates elements by using qualified namespaces. With this structure in place, we can easily direct our documents to use the right elements anywhere in the enterprise. A good example would be using a BMA element inside the WMA structure: (the only rule is, the element must be of the same data type and formats; we will cover data types in the schema section)

```
<wma:person>
        <bma:image>1110010101</wma:image>
        <wma:Name>Joe Smith</wma:Name>
        <wma:SSN>111-11-1111</wma:SSN>
        <wma:Unit>A Co, 2/34th</wma:Unit>
</wma:person >
```

Army SOA XML Data Strategy # 1

In order to standardize namespace resolutions, the Army CIO/G-6, Army Architecture Integration Cell (AAIC) is defining four (4) parent namespace scopes within the SOA foundation architecture. These namespace scopes will be the authoritative name attributes to be used inside the Foundation. The following structure is proposed:

1. Four Mission Area authoritative Namespaces for NIPR environment:

   - http://soa.army.mil/schemas/wma
   - http://soa.army.mil/schemas/bma
   - http://soa.army.mil/schemas/ima
   - http://soa.army.mil/schemas/eiema

Each namespace will have authoritative sub-namespaces, these namespaces will be governed and certified by the Army SOA Data certification group, chartered by the CIO/G-6. A web presence will be maintained for each parent namespace and a list of all available sub-namespaces will be displayed when an entity navigates to a parent namespace location. For example:

http://soa.army.mil/schemas/wma will display sub-namespaces belonging to the WMA parent namespace:

1. http://soa.army.mil/schemas/wma/jc3iedm
2. http://soa.army.mil/schemas/wma/gml
3. http://soa.army.mil/schemas/wma/bft .....

A representation of the BMA namespace may have sub-namespaces as follows:

1. http://soa.army.mil/schemas/bma/bpel
2. http://soa.army.mil/schemas/bma/log
3. http://soa.army.mil/schemas/bma/hr .....

An automated mechanism will be in place for validation and certification of sub-namespaces. The certifying entity will ensure namespaces are also register with DISA's namespace registry.

2. Four Mission Area authoritative Namespaces for SIPR environment:

- http://soa.army.smil.mil/schemas/wma
- http://soa.army.smil.mil/schemas/bma
- http://soa.army.smil.mil/schemas/ima
- http://soa.army.smil.mil/schemas/eiema

The same sub-namespace structure will be available for the SIPR implementation. Both, NIPR and SIPR namespace structure should maintain the same sub-namespaces for compatibility.

3. For disconnected and enclave solutions a file structure will be used for both the NIPR and the SIPR. The file structure will be an exact representation of the web presence. The namespace folders will maintain the sub-namespaces as folder locations within the file system. See *figure 4*, for an example.

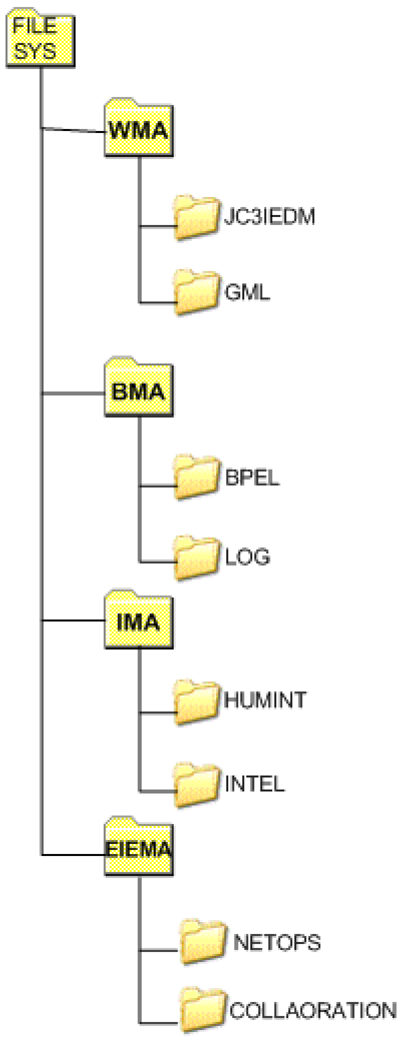


Figure 3, Namespace File structure

With these namespace structures in place, data models will no longer drive data strategy. XML modularity will allow for multiple models to be used within one XML document (payload), while data models continue to provide their intended benefits; data definition. The next section will build upon XML namespaces by providing the mechanism that ensures our XML data is compliant to the specific namespaces that owns it; this mechanism is the XML Schema.

XML Data Schemas

XML Schema is a schema-definition language expressed in XML syntax. To avoid ambiguity, XML Schema is often referred to as XSD Schema because in an earlier version it was called XML Schema Definition Language. Recently, the abbreviation WXS has also come into use to refer to the XML Schema language. An XML Schema describes the structure of an XML document (payload).

The purpose of an XML Schema is to define the legal building blocks of an XML document (payload).

An XML Schema:

- defines elements that can appear in a XML document (payload)
- defines attributes that can appear in a XML document (payload)
- defines which elements are child elements
- defines the order of child elements
- defines the number of child elements
- defines whether an element is empty or can include text
- defines data types for elements and attributes
- defines default and fixed values for elements and attributes

Schemas and the communication layer

When sending data from a sender to a receiver, it is essential that both parties have the same "expectations" about the XML payload. With XML Schemas, the sender can describe the data in a way that the receiver (Web services) will understand.

A date like: "05-11-2007" will, in some instances, be interpreted as 5 November 2007 and in other as 11 May 2007. However, XML elements with a data type like this:

```
<date type="date">2007-05-11</date>
```

ensures a mutual understanding of the content, because the XML data type "date" requires the format "YYYY-MM-DD".

XML Schemas are extensible, because they are written in XML. With an extensible Schema definition you can:

- Reuse your Schema in other Schemas
- Create your own data types derived from the standard types
- Reference multiple schemas in the same document

XML schemas are used to validate XML documents (payload). With this in mind, we can build on the previous *person* xml document (payload), but instead of using the WMA as the namespace, let's use a sub-namespace below the WMA – JC3IEDM as a namespace which is the authoritative owner of the *person* data structure. Also, assume that we will save the xml document to a file called "*jc3iedm_sample.xml*"

```
<?xml version="1.0"?>
<JC3IEDM:person xmlns:JC3IEDM=http://soa.army.mil/schemas/wma/jc3iedm
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http:// soa.army.mil/schemas/wma/jc3iedm  jc3iedm.xsd

        <JC3IEDM:image>1110010101</JC3IEDM:image>
        <JC3IEDM:Name>Joe Smith</JC3IEDM:Name>
        <JC3IEDM:SSN>111-11-1111</JC3IEDM:SSN>
        <JC3IEDM:Unit>A Co, 2/34th</JC3IEDM:Unit>
</JC3IEDM:person>
```

By now the only lines that should not be familiar are the ones in blue. The lines in blue add a reference to an XML Schema.

Once the XML Schema Instance namespace is available:

```
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
```

Data maodelers can use the schemaLocation attribute, which is an element that belongs in the www.w3.org namespace. This attribute has two values. The first value is the namespace to use. The second value is the location of the XML schema to use for that namespace:

```
xsi:schemaLocation=" http:// soa.army.mil/schemas/wma/jc3iedm  jc3iedm.xsd"
```

Identifying the schema location is like telling the application (XML parser), go to the web location http://soa.army.mil/schemas/wma/jc3iedm/jc3iedm.xsd and pull down the schema file (XSD) and make sure the elements in the file named *jc3iedm_sample.xml* are behaving the way they supposed to, if not, send and error message telling the application the XML document (payload) is not valid. The power of schemas is simple; they make the data types (or elements) authoritative across usage scenarios.

Creating the schema

Creating the schema document (XSD) is pretty close to creating an XML document. Here we create the *jc3iedm.xsd* schema that will validate the *jc3iedm_sample.xml* document.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://soa.army.mil/schemas/wma/jc3iedm"
      xmlns=" http://soa.army.mil/schemas/wma/jc3iedm ">


     <xsd:element name="person"/>
       <xsd:complexType>
            <xsd:sequence>
                <xsd:element name="image" type="xsd:base64Binary" />
                <xsd:element name="Name" type="xsd:string" />
                <xsd:element name="SSN" type="xsd:string" />
                <xsd:element name="Unit" type="xsd:string" />
            </xsd:sequence>
     </xsd:complexType>
   </xs:element>
</xsd:schema>
```

Again, all elements that are preceded by the xsd belong to the www.w3.org namespace. We used xsd as the prefix, but we could have used any other word.

The XML Schema document (person, is the root) element is declared as follows:

```
<xsd:element name="person"/>
```

To find the allowed content of the *person* element, you need to find the definition of the complexType below the *person* definition:

```
<xsd:complexType>
          <xsd:sequence>
                <xsd:element name="image" type="xsd:base64Binary" />
                <xsd:element name="Name" type="xsd:string" />
                <xsd:element name="SSN" type="xsd:string" />
                <xsd:element name="Unit" type="xsd:string" />
          </xsd:sequence>
</xsd:complexType>
```

This definition specifies that a *person* element is allowed to have a sequence of child elements (image, Name, SSN and Unit), as defined by the content of the xsd:sequence element.

The *person* element is a **complex type** because it contains other elements. The other elements (image, Name, SSN and Unit) are **simple types** because they do not contain other elements.

The values of some elements are simply strings, as indicated by the xsd:string value for the type attribute. The xsd:string type is one of many built-in datatypes in W3C XML Schema.

This may all seem a little complicated, but it is key for validation: if the jc3iedm_sample.xml document is opened by a web browser or a web service, the browser/service will automatically pull the jc3iedm.xsd from its location and ensure the elements: *image* is of type binary, *Name* is of type string, *SSN* is of type string and *Unit* is of type string. If the elements in the jc3iedm_sample.xml document do not comply with these rules, the browser/service will send a not validated message.

Army SOA XML Data Strategy # 2

By now it is very obvious namespaces and schemas work hand-in-hand. As in Strategy #1, we must standardize the locations these schemas will be positioned. The following structure is proposed:

The namespace parent structure will be used in combination with schemas locations:

- http://soa.army.mil/schemas/wma
- http://soa.army.mil/schemas/bma
- http://soa.army.mil/schemas/ima
- http://soa.army.mil/schemas/eiema

For each parent location and sub-namespace location (in NIPR, SIPR and File Structure), a specific XML schema definition will be provided. For example:

4. http://soa.army.mil/schemas/wma/jc3iedm/jc3iedm.xsd

5. http://soa.army.mil/schemas/wma/gml/gml.xsd
6. http://soa.army.mil/schemas/wma/bft/bft.xsd .....

A representation of the BMA schemas may be represented as follows:

4. http://soa.army.mil/schemas/bma/bpel/bpel.xsd
5. http://soa.army.mil/schemas/bma/log/log.xsd
6. http://soa.army.mil/schemas/bma/hr/hr.xsd .....

A representation of the file schema structure will be as follows:

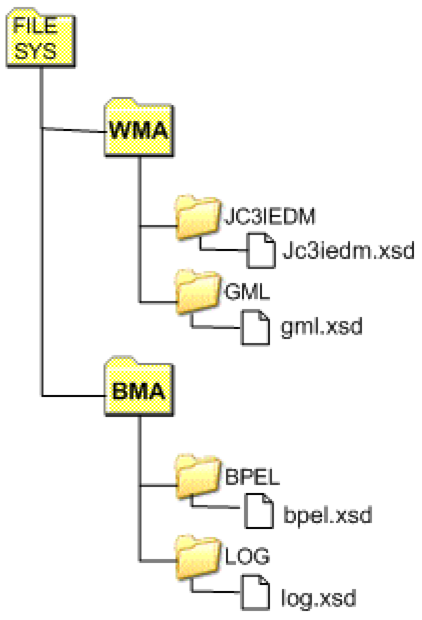


Figure 4, xsd schema structure

It is now valid to say "Communities of Interest (COIs) will dictate the structures of many XSDs." It is up to the Army's SOA Data certification group to work with COIs in order to deploy the schemas the COIs agree will be authoritative within their sphere of influence.

Now that we have defined the namespace and schema structures, we will see how we can leverage these structures in a SOA environment. The next sections will cover how WSDL and SOAP use namespaces and schemas to ensure a standard data validation in the Enterprise.

On its most fundamental level, SOA is built upon and driven by XML and its standards. As a result, an adoption of SOA leads to the opportunity to fully leverage the XML data paradigm. A standardized data representation format (once fully established) can reduce the underlying complexity of all affected application and document environments.

Some examples include:

- XML documents (payloads) and accompanying XML Schemas (packaged within SOAP messages) passed between applications or application components fully standardized. The result is a predictable and therefore easily extendible and adaptable communication network.
- XML's self-descriptive nature enhances the ability for data to be readily interpreted by architects, analysts and developers and increases the potential for data within messages to be more easily maintained, traced and understood.
- The standardization level of data representation lays the groundwork for intrinsic interoperability. Specifically, by promoting the use of standardized vocabularies (schemas); the need to translate discrepancies between how respective applications have interpreted Army data models is reduced.

By defining a common structure for namespace and schemas, the Army can fully take advantage of the SOA paradigm and adjust to architecture changes in the future.

Web Services Description language (WSDL)

Wikipedia describes WSDL as "*an XML-based service description on how to communicate using web services. The WSDL defines services as collections of network endpoints, or ports.*

WSDL is often used in combination with SOAP and XML Schema to provide web services over the Internet. A client program connecting to a web service can read the WSDL to determine what functions are available on the server. Any special data types used are embedded in the WSDL file in the form of XML Schema. The client can then use SOAP to actually call one of the functions listed in the WSDL."

We will not try to cover the vast aspects of WSDL in this document. We will focus on those items relating to the namespace and schemas.

WSDL Basics

One of SOA's key ingredients is to establish a consistent loosely coupled form of communication between services implemented as Web services. For this purpose, description documents are required to accompany any Web service wanting to act as ultimate receiver. The primary service description document is the WSDL definition (Figure 5).

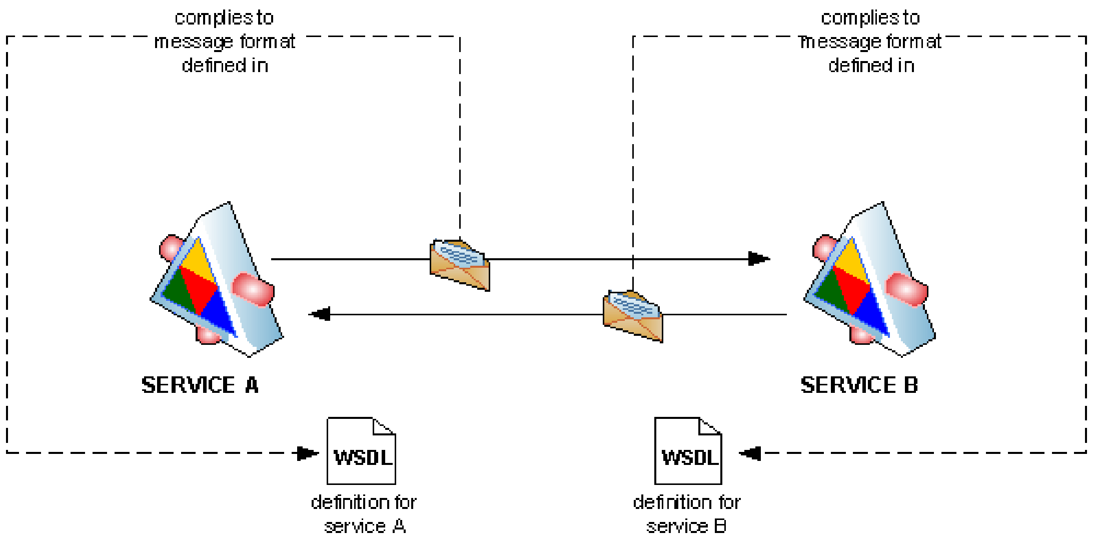

complies to message format defined in
complies to message format defined in
SERVICE A
SERVICE B
WSDL
WSDL
definition for service A
definition for service B

Figure 5. definitions enable loose coupling between services

A WSDL describes the point of entry for a service provider, also known as a service endpoint. It provides a formal definition of the endpoint interface (so that requestors wishing to communicate with the service provider know exactly how to structure request messages) and also establishes the physical location (address) of the service.

For the purpose of brevity, let's break WSDL service definition into two categories:

- Abstract description
- Concrete description

Abstract description:

Establishes the interface characteristics of the Web service without any reference to the technology used to host or enable a Web service to transmit messages.

Concrete description:

For a Web service to be able to execute any of its logic, it needs for its abstract interface definition to be connected to some real, implemented technology. Because the execution of services application logic always involves communication, the abstract Web service interface needs to be connected to a physical transport protocol. This connection is defined in the concrete description.

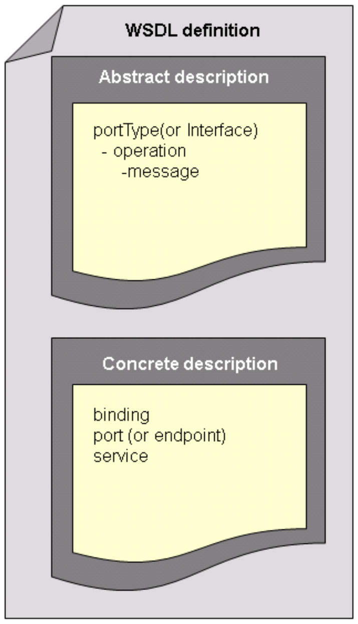


Figure 6, WSDL Document definition

For the purpose of this document, we will only focus on the abstract description portion in this section. The SOAP section will address the concrete description part of the WSDL definition.

The abstract definition contains a series of parts that include types, message and port type (or interface), whereas the concrete definition is comprise of binding and service parts.

Each of these parts relates to corresponding elements that are defined in the WSDL specification.

WSDL Definition Element

The definition element is the root or parent element of every WSDL document. It houses all other parts of the service definition and is also the location in which the namespaces used within the WSDL document are established.

```
<definitions name=”Person”>
     targetNamespace="http://soa.army.mil/schemas/wma/wsdl"
     xmlns=" http://schemas.xmlsoap.org/wsdl"
     xmlns :jc3iedm=" http://soa.army.mil/schemas/wma/jc3iedm"
     xmlns :wma=" http://soa.army.mil/schemas/wma"
     xmlns :bma=" http://soa.army.mil/schemas/bma"
</definitions>
```

In the preceding example, the service definition is started with a *definition* element that contains a series of attributes in which the service is assigned the name of “Person” and in which a number of namespaces are declared.

WSDL Types Element

The *types* construct is where XSD schema content is placed. This part of the WSDL can consist of actual XSD schema markup (an entire schema construct containing type definitions; like our previous jc3iedm.xsd), or it can contain import or include elements that reference external schema definitions.

```
<types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     targetNamespace="http://soa.army.mil/schemas/wma/jc3iedm"
     xmlns=" http://soa.army.mil/schemas/wma/jc3iedm ">


    <xsd:element name="person"/>
      <xsd:complexType>
           <xsd:sequence>
               <xsd:element name="image" type="xsd:base64Binary" />
               <xsd:element name="Name" type="xsd:string" />
               <xsd:element name="SSN" type="xsd:string" />
               <xsd:element name="Unit" type="xsd:string" />
           </xsd:sequence>
    </xsd:complexType>
   </xsd:element>
   </xsd:schema>
<types>
```

The only thing new in the schema above is the *type* element inside the WSDL document. Even though we can construct an entire schema inside the WSDL, it becomes very unmanageable when dealing with big XSD schema definitions. A better solution would be to import or include the schemas from an external source like the sample below.

```
<types>
     <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
     <import schemaLocation="http://soa.army.mil/schemas/wma/jc3iedm/jc3iedm.xsd"
            namespace="http://soa.army.mil/schemas/wma/jc3iedm"/>
     <include schemaLocation=http://soa.army.mil/schemas/bma/bpel/bpel.xsd"/>
</types>
```

XSD schemas can be modularized. This allows for one schema document to import the contents of another. Both *import* and *include* elements are used to point to the location of the XSD schema file that will be pulled in when the schema is processed at runtime. The above example pulls the *jc3iedm.xsd* from the WMA JC3IEDM namespace and *bpel.xsd* from the BMA BPEL namespace.

The difference between import and include is that *include* is used to reference schemas that use the same target namespace as the parent schema, whereas *import* is used to point to schemas that use a different target namespace.

WSDL Message and Part Element

For every message a service is designed to receive or transmit, a message construct must be added. This element assigns the message a name and contains one or more part child elements that each are assigned a type.

Part elements use the *type* or element attributes to identify the data type of the message part. The type attribute can be assigned a simple or complex type and generally is used for RPC-style messages. Part elements in document-style messages typically rely on the element attribute, which can reference an XSD element.

```
<message name="PersonRequestMessage">
     <part name="RequestParameter"
            Element="JC3IEDM:SSN"/>
</message>

<message name="PersonResponseMessage">
     <part name="ResponseParameter"
            Element="JC3IEDM:person"/>
</message>
```

In the example above two messages are being exchanged through the Web service, one is a request and the other a response. The first is a message called "*PersonRequestMessage*" and attached to this message is a validated XSD element from the JC3IEDM namespace and schema (jc3iedm.xsd). The JC3IEDM:SSN is a request being pass as input to a web service.

The second message is the response to the request from the first message. This is evident by the "ResponseParameter" element and the type of data type it is expecting: JC3IEDM:person. Remember, the XSD schema describes a person as an image, Name, SSN and Unit. Therefore the response payload will be an XML document similar to the one below.

```
<?xml version="1.0"?>
<JC3IEDM:person xmlns:JC3IEDM=http://soa.army.mil/schemas/wma/jc3iedm
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http:// soa.army.mil/schemas/wma/jc3iedm  jc3iedm.xsd

        <JC3IEDM:image>1110010101</JC3IEDM:image>
        <JC3IEDM:Name>Joe Smith</JC3IEDM:Name>
        <JC3IEDM:SSN>111-11-1111</JC3IEDM:SSN>
         <JC3IEDM:Unit>A Co, 2/34th</JC3IEDM:Unit>
</JC3IEDM:person>
```

Seems familiar – it's the same XML document (payload) we started with at the beginning of this paper.

Army SOA XML Data Strategy # 3

WSDL is an XML-based service description on how to communicate using web services. In order to standardize how we represent WSDL and its constructs' in the Army Enterprise, there are a few things we must ensure.

The following WSDL standardization is proposed:

- WSDL *definition* section for namespace naming convention will be constructed with the same namespace naming conventions as the proposed Army SOA Data Strategy #1 structure.
- The *type* elements definition within the WSDL will use the *import* and *include* attribute constructs and the namespaces will point to the namespace naming

convention as SOA Strategy #1. The schema (XSD) within the *type* definition will always point to a schema structure as outline in the proposed Army SOA strategy # 2.

- Schema Data structures will not be directly entered into the WSDL document.
- The message definition portion of the WSDL will only use document-style messages which rely on the *element* attribute. The RPC-style type attribute will not be used within a WSDL document.
- Auto-generated WSDL documents will not be supported under the Army SOA Data Strategy. Every WSDL will follow standardize naming (namespace) and schema definition.

The transport Layer- SOAP

SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP. SOAP forms the foundation layer of the Web services stack, providing a basic messaging framework so that more abstract layers can build on.

SOAP originally stood for Simple Object Access Protocol, and lately also Service Oriented Architecture Protocol, but is now simply SOAP. The original acronym was dropped with Version 1.2 of the standard, which became a W3C Recommendation on June 24, 2003, as it was considered to be misleading.

SOAP Basics

In the previous section we established that a WSDL definition intentionally separates abstract from concrete definition details. One of the benefits of doing so is that we can isolate communication protocols that implements the messaging required by a service from the implementation-neutral service interface. Given that SOAP has become the messaging format of choice for SOA, we will very likely be binding the abstract interface of the WSDL to the SOAP protocol.

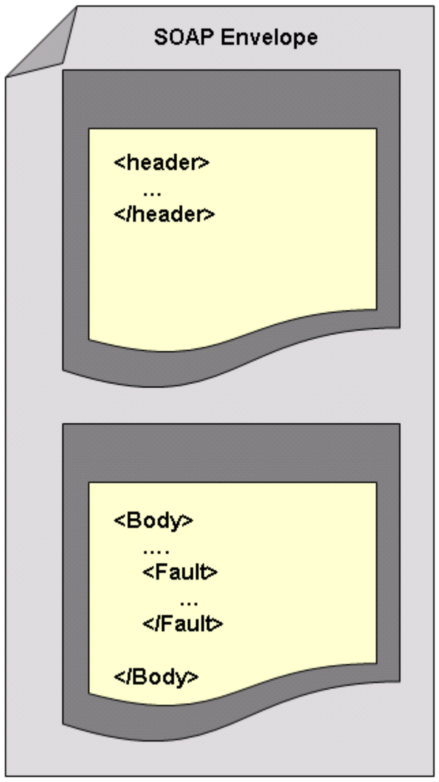


Figure 7, SOAP message structure

The structure of SOAP messages consists of header, body and fault sections, all encased in an enveloped (See figure 7).

For the purpose of brevity, this paper only covers the *body element* of the SOAP document. The *Envelope* element is the root of the SOAP message. It contains a mandatory *Body* element and an optional *Header* element.

The *Header* portion of the SOAP message has become a key enabler of the feature set provided by the WS-*

specifications. WS-Policies, WS-Security and reliable messaging are actualized through the SOAP Header construct.

SOAP Body Element

The *Body* element is a mandatory child element of the SOAP *Envelope* element. It contains the message payload formatted as well-formed XML data.

SOAP message Body constructs are defined within the WSDL message constructs which, as we've already established, reference XSD schema data type information from the WSDL types construct (Figure 8).

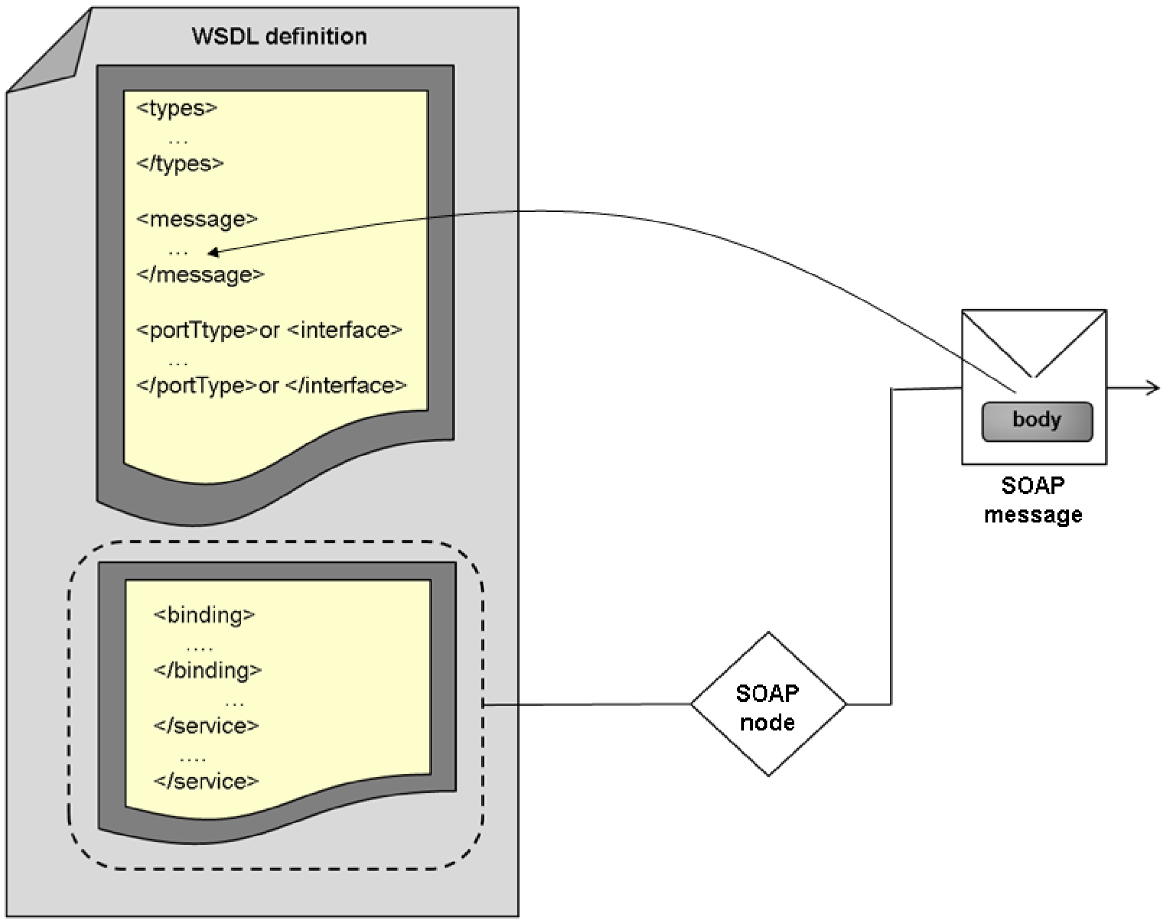


Figure 8, A SOAP message body defined within the WSDL message element

From the figure above, it is apparent why SOAP couples so well with WSDL. SOAP is the transport mechanism for web services and it's defined through XML standards. Add data checking (XSD Schemas) and strong naming conventions (namespaces), and we have a reliable an integrated communication mechanism for documents, applications, services and data.

Here is an example of how a client might format SOAP message requesting *person* information from a web service that may reside in the WMA and using the JC3IEDM namespace and schema. The client knows a soldier's Social Security Number (SSN – 111-11-1111), the request would look like:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    < PersonRequestMessage xmlns=" http://soa.army.mil/schemas/wma/jc3iedm">
            <SSN>111-11-1111</SSN>
    </ PersonRequestMessage >
  </soap:Body>
 </soap:Envelope>
```

And here is a possible response to the client request:

```
<soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <soap:Body>
    < PersonResponseMessage xmlns=" http://soa.army.mil/schemas/wma/jc3iedm">
            <person>
                 <image>1110010101</image>
                 <Name>Joe Smith</Name>
                 <SSN>111-11-1111</SSN>
                 <Unit>A Co, 2/34th</Unit>
           </person>
    </ PersonResponseMessage >
  </soap:Body>
 </soap:Envelope>
```

One final point on SOAP; SOAP is versatile enough to allow for the use of different transport protocols. The standard stacks use HTTP as a transport protocol, but other protocols are also usable (TCP, SNMP).

Summary

The coupling of Namespace, Schemas (XSD), WSDL and SOAP is proving to be the driving force behind SOA. The Army faces many challenges as it starts positioning itself to take advantage of this new architecture.

One of the first steps we must take is to standardize how XML data is represented within the Army SOA foundation and the Enterprise:

1. SOA XML Data Strategy # 1 – Namespace consistent structures
2. SOA XML Data Strategy # 2 – Schema consistent location structure
3. SOA XML Data Strategy # 3 – Standardizing WSDL abstract description

Hopefully, the content within this document addresses some of the key XML Data challenges over the horizon. Standardizing on our core (Parent) namespaces and

XSD schemas for our top level Mission Areas and by addressing consistent formats for the WSDL and SOAP constructs we will be well position to take advantage of this new technology.